

Einführung

- Data Warehouse:
 - Die Daten werden regelmäßig von der Produktivdatenbank abgefragt
 - Daten werden zum Lesen vorbereitet
 - Daten auf Dauer speichern → Archiv
 - erlaubt Fragen mit einer breiteren Antwortmöglichkeit (z.B. Kunden die x kauften, kauften auch y)
 - Vorteile:
 - unabhängig von der Datenbank
 - keine zusätzliche Last für die Datenbank
 - Direkter OLAP Zugriff von mehreren Usern stresst nur DW und nicht Datenbank
 - Nachteile:
 - Eigenes System zum Verwalten
 - Data Warehouse muss geupdated werden, meistens über Nacht
- **Gründe für DW:**
 - Keine zusätzliche Last auf Produktivsystem
 - Daten werden für Jahre gespeichert zum vergleichen, diese sind nicht im Produktivsystem vorhanden
 - Produktivsystem ist für Transaktionen optimiert und erlaubt keine Redundanz

	Productive database	Data Warehouse
Aim	Transaction processing (Geschäftsabwicklung)	Analytical processing (Geschäftsanalyse)
Important	Detailed data	Aggregated data
Data pool	Current data only	The history of all the data
DB size	~ 10 GB – 1 TB	~ 100 GB – 10 TB
DB access	Updates and queries	Only queries
Accesses/operation	~ 10 data records	~ 1.000.000 data records
Performance	Optimized for throughput	Optimized for response time

- **OLTP: Online Transaction Processing**
 - Produktivsystem mit Transaktionen
 - Transaktionssicherheit, Hoher Durchsatz, Kurze Antwortzeit, Paralleles Verarbeiten
- **OLAP: Online Analytical Processing (→ Data Warehouse)**
 - nur zum Auslesen, Analyse der Daten im Vordergrund
- **FASMI: Fast Analysis of Shader Multidimensional Information**
 - Fast: die meisten Antworten kommen innerhalb von 5 Sekunden
 - Analysis: Analyse ist einfach genug für den Zieluser
 - Shared: Gleichzeitige Schreib- & Lesezugriffe können verwaltet werden
 - Multidimensional: Multidimensionale, konzeptionelle Sicht auf die Daten
 - Information: System verwaltet alle Eingabedaten und speichert diese

Datenbank zu Data Warehouse

- Characteristics: Merkmale
 - o Basisdaten, die für Abfragen verwendet werden (z. B. Produkt, Kunde, Datum)
- Key figures: Kennzahlen
 - o Resultate, die für Basisdaten erwartet werden, können sich in DB ändern, aber werden in DW festgehalten
 - z.B. Verkaufspreis, Anzahl, Wachstum ...
- Fact Table: optimiert Datenzugriff über verschiedene Tabellen
 - o beinhaltet Characteristics (OrderID, CustID, ArtID, Salesdate), Key figures (Quantity, Price) und Zusatzinformationen (Monat, Jahr)
 - o meistens Join von verschiedenen Tabellen
 - o Vorteile:
 - Ein Join weniger
 - Optimierung ist unabhängig von Datenbank
 - Anfragen brauchen keinen Zugriff auf einzelne Tabellen
 - o Nachteile:
 - Zusätzliche Tabelle (Verwaltung, Laden der aktuellen Daten)
 - o Aggregationen:
 - Daten werden in einzelnen Tabellen angezeigt, z.B. pro Monat
- Materialized View: aktualisiert Daten nebenbei, um gleich den Originaldaten zu sein
 - o wird mit Daten nach dem „START WITH“ date befüllt
 - o wird upgedated mit dem „NEXT“ date

```
CREATE MATERIALIZED VIEW Viewname [ (Columnlist) ]
[ BUILD IMMEDIATE | BUILD DEFERRED ]
[ REFRESH [ FAST | COMPLETE ] [ START WITH Date ] [ NEXT Date ] ]
AS Select-command
```

› BUILD IMMEDIATE	created immediately
› BUILD DEFERRED	created with the first refresh
› REFRESH FAST	transfers the data changed only
› REFRESH COMPLETE	transfers all data
› REFRESH START WITH	date of first transfer
› REFRESH NEXT	date of next and further transfers

Daten in Datenbank

- Master Data: Alle Informationen, die eine wichtige Rolle in der Ausführung eines Geschäfts spielen
 - o z.B. customers, supplier, region, country, articles, stock location ...
- Transactional Data: Datensatz, der unvorhersehbar und zufällig während der Ausführung ist
 - o z.B. order, account, production, sale, purchase

Beispiel INSERT INTO Facttab_Month mit Group by:

```
INSERT INTO fact1_month (mid, artid, month, year, price, quantity)
```

```
SELECT
```

```
mid_counter.NEXTVAL,
FM.*
```

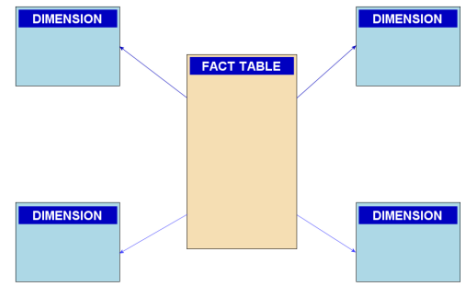
```
FROM
```

```
( SELECT artid, month, year, round(AVG(price), 2), SUM(quantity)
  FROM fact1           //mit JOIN ... USING (SSID)
 GROUP BY artid, month, year ) as FM;
```

Konstruktion eines Data Warehouse

- Star Model:

- o Faktttabelle, umgeben von Dimensionen (Master Data)
- o Faktttabelle beinhaltet IDs und Key Figures



- Snowflake Model

- o Faktttabelle, umgeben von Dimensionen, mit zusätzlichen Dimensionen
- o Weniger Redundanz, weil Dimensionen mit IDs auf Daten zugreifen (z.B. ID Day of Week...)

- Multi Dimensional Model

- o Faktttabelle mit Dimensionen ist ein „Cube“, mehrere Cubes miteinander sind eine Galaxy

zum Star Model:

- ROLAP:

Relational OLAP

- o Daten werden relational gespeichert, Faktttabelle und Dimensionen sind Relationen, verbunden durch FK
- o Vorteile:
 - Existierende, high-performance Relationale Datenbanken können benutzt werden
 - Wir können SQL zum abfragen nutzen
- o Nachteile:
 - viele joins → Performance Probleme:
 - Star Schema mit n Dimensionen braucht n Joins
 - wir brauchen zusätzliche, High-Performance SQL-Operatoren

- MOLAP:

Multidimensional OLAP

- o fixiertes und statisches, multidimensionales Array (keine Datenbank), Einträge zu allen Kombinationen von Characteristics
- o Vorteile:
 - wenige Joins, sehr performant
- o Nachteile:
 - proprietäres System, ungewöhnliche Vorgehensweise für Abfragen, viel Speicherverbrauch
- o nur für kleine Würfel geeignet

- HOLAP:

Hybrid OLAP

- o Mix zwischen ROLAP und MOLAP, benutzt Vorteile beider Modelle
 - Große Würfel in ROLAP, kleine Würfel und aggregierte Würfel mit in MOLAP

- Oracle:

ROLAP, zusätzlich zu memory datenbanken

- MS SQL Server:

MOLAP ist zusätzlich implementiert

- SAP BW on HANA:

ROLAP mit vielen Spalten und vielen Speichertechniken

Abfragen:

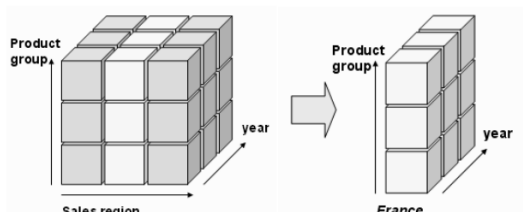
- Definitionen:

- o Drill Down: von gesammelten Informationen zu Detail durch Fokus auf ein Detail
- o Roll Up: Gegenteil von Drill Down, Details werden zu gesammelten Informationen aggregiert

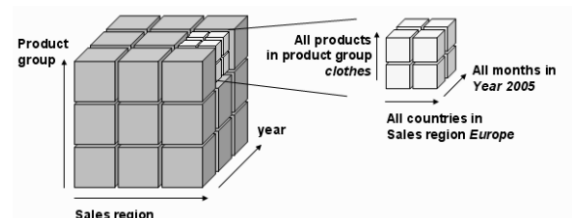
- Slicing und Dicing:

- o Slicing: Scheibe aus einem Würfel schneiden, also nur eine Bedingung
- o Dicing: kleinen Würfel aus dem großen Würfel mit gleichen Dimensionen schneiden

Slicing:



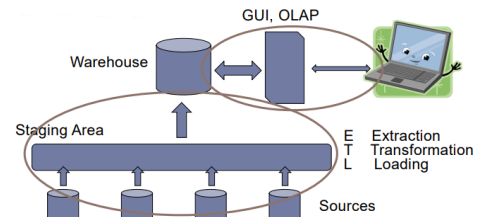
Dicing:



ETL-Prozess

- **ETL-Prozess** Extraction, Transforming, Loading
 - o Daten werden in Staging Area geladen, transformiert & persistiert, erst zu günstiger Zeit werden Daten in Data Warehouse geladen
 - o Sources:
 - Relationale Datenbanken, Daten sollen am besten ohne Redundanz geladen werden
 - o Staging Area:
 - Interface für die Sources (meist relationale DB)
 - persistent, weil Laden des Warehouses fehlschlagen kann
 - Sicheres Interface zwischen OLTB DB und Warehouse
 - o Warehouse
 - NoSQL Datenbank oder proprietäres System
 - Performance ist extrem wichtig
 - Redundanz ist nicht so wichtig

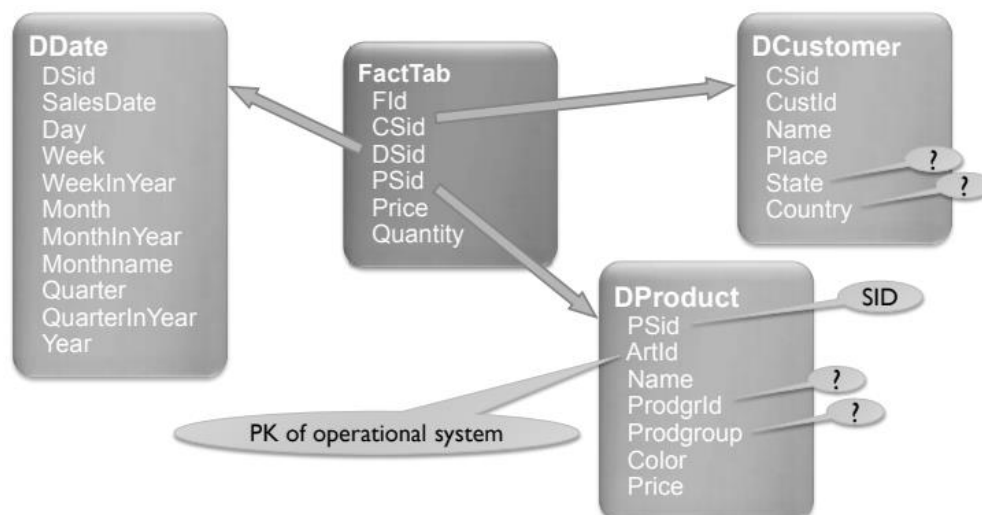
Data Flow



- Extraction
 - o Staging Area: die externen Daten werden für das Data Warehouse vorbereitet
 - Quellen: aus Datenbanken, Excel-Files, Text-Files, ...
- Transformation
 - o Transformation der Daten in der Staging Area
 - o Datentypen adaptieren, konvertieren von speziell gecodeten Daten (ASCII, Unicode), unifying of Strings and Dates, Messeinheiten konvertieren
 - o Probleme: Extrahierte Daten können out of Date, inkorrekt, inkonsistent, unvollständig sein
- Loading
 - o Laden der Daten aus der Staging Area in Data Warehouse (erhöht response time)
 - o Laden während einer Zeit mit wenig Anfragen, z.B. Nachts
 - o Abfragen werden limitiert während dem Laden

GUI:

- soll Abfragen auf alle Daten ermöglichen
- Einfach und intuitiv
- Technische Details werden versteckt
- Automatische Erstellung von Instruktionen mit hoher Performanz
- Zugriff für Idioten die kein SQL können soll möglichst einfach sein



Arbeiten mit dem Data Warehouse

Star Join:

- 3. Normalform: Redundanz wird verhindert durch Snowflake Modell
- Join mit Join mit Join ... → durch IDs wird von einer Tabelle zur nächsten gegangen
- Surrogat Key:
 - o Enumeration
 - o z.B. Csid → Customer Surrogat ID: Primärschlüssel in den Dimensionen
- Join zwischen Dimensionen und Fakttabellen

ON, USING, NATURAL

```

SELECT Sum ( F.Quantity )
FROM Customer C JOIN Fact F ON F.CSid = C.CSid
JOIN Date D ON F.DSid = D.DSid
JOIN Product P ON F.PSid = P.PSid
WHERE C.Country = 'Germany'
AND D.Month = 10 AND D.Year = 2019
AND P.Category = 'Electrical appliance' ;

```

Outer Join:

- Vorziehen von SELECT-Befehlen in den FROM-Teil: Ergebnis wird erst gefiltert und dann gejoined, dadurch kann man auch Ergebnisse anzeigen, die bestimmte Filter sonst nicht erfüllen

```

SELECT ArtId, Name, COALESCE( SUM( Quantity ), 0 ) AS Quantity
FROM ( SELECT DSid FROM DDate WHERE Month=201902 )
INNER JOIN Facttab USING (DSid)
INNER JOIN ( SELECT CSid FROM DCustomer WHERE State = 'Bayern' ) USING (CSid)
RIGHT OUTER JOIN
( SELECT * FROM DProduct WHERE Prodgroup = 'Lady bicycle' ) USING (PSid)
GROUP BY ArtId, Name
ORDER BY ArtId;

```

Cross Join:

- A **PARTITION BY** (Attribut1, Attribut2) **RIGHT OUTER JOIN** B ON ...
 - o Attribute in Partition By: diese sollen gecrossed werden mit B
 - alle spezifizierten Attribute werden einmal mit komplett B gekreuzt

Join	Effect
A Inner Join B On ...	The data found in both relations A and B are displayed.
A Right Outer Join B On ...	The data found in both relations A and B are displayed included by the remaining data from B. In the latter case the corresponding attributes of A are set to null.
A Partition by (attributes) Right Outer Join B On ...	All data of the specified attributes from A are displayed. All data from B are joined with them row by row and displayed. The not corresponding data are set to null. all from A! for each A: all from B!
A Full Outer Join B On ...	All data found in both relations A and B are displayed included by the remaining data from A and from B. The not corresponding data from A and B are set to null.

- A **RIGHT OUTER JOIN** (SELECT B **CROSS JOIN** C) **USING** ...
 - o **CROSS JOIN**: alle Zeilen von B mit allen Zeilen von C kreuzen
 - o Reihenfolge:
 1. INNER JOINS
 2. RIGHT OUTER JOIN
 3. CROSS JOIN
 4. 2 USINGs nach CROSS JOIN → sind ja 2 Tabellen die gekreuzt werden

```

SELECT C.CustId, C.Name As Customer, P.ArtId, P.Name As Article,
       COALESCE(SUM( Quantity ), 0) AS Quantity
FROM ( SELECT DSid FROM DDate D WHERE Month=201902 )
INNER JOIN Facttab F USING (DSid)
RIGHT OUTER JOIN
( ( SELECT * FROM DProduct WHERE Prodgroup = 'Lady bicycle' ) P
  CROSS JOIN
  ( SELECT * FROM DCustomer WHERE State = 'Bayern' ) C )
USING (PSid, CSid)
GROUP BY C.CustId, C.Name, P.ArtId, P.Name
ORDER BY C.CustId, P.ArtId;

```

Query Rewrite (Materialized View):

- Materialized View kann mit der Option „ENABLE QUERY REWRITE“ erstellt werden
- Dadurch nutzt Oracle automatisch die MV, wenn es sich anbietet, um die Performance zu verbessern
 - o auch wenn eigentlich auf die Facttab zugegriffen wird, wenn die Option Rewrite eingeschaltet ist, optimiert Oracle automatisch die Abfrage

```
CREATE MATERIALIZED VIEW Facttab_Month (PSid, Name, MSid, SalesVolume, Quantity)
ENABLE QUERY REWRITE
AS SELECT /*+ MV_MERGE */ -- Hint is necessary in Oracle >= V12
      PSid, Name, Month, SUM(Facttab.Price*Quantity), SUM(Quantity)
FROM   Facttab INNER JOIN DDate USING(DSid)
      INNER JOIN DProduct USING(PSid)
GROUP BY PSid, Name, Month;
```

```
SELECT /*+REWRITE*/ PSid, Name, Month, SUM(FPrice*Quantity) As SalesVolume,
      SUM(Quantity) As Quantity
FROM   Facttab F INNER JOIN DDate D USING (DSid)
      INNER JOIN DProduct P USING (PSid)
GROUP BY PSid, Name, Month ;
```

Dimensions:

- CREATE DIMENSION stellt Hierarchie wieder her (vgl. DDate: Jahr, Quartal, Monat, Tag)
- sonst nimmt QUERY REWRITE für z.B. Jahr nicht die erstellte MV her, sondern berechnet es einzeln → Performanceeinbußen
 - o LEVEL: Hierarchie Level werden bezeichnet und mit den Spalten verknüpft
 - o HIERARCHY: Beschreibung der Hierarchie: *m to n* Relation (Foreign Keys von Tabellen der 3. NF)
 - o ATTRIBUTE: Beschreibung von zusammenhängenden Spalten innerhalb der Tabelle
- Problem: Im Star Model wird die 3. Normalform aufgegeben
 - o Hierarchien existieren nicht mehr, Oracle braucht Anweisungen
- Validierung: EXECUTE DBMS_DIMENSION.VALIDATE_DIMENSION(,DATE_DIM', FALSE, TRUE)
 - o Nur nach Validierung wird Dimension mit Query Rewrite benutzt

Create Dimension Date_Dim

```
LEVEL Day IS DDate.SalesDate
LEVEL Week IS DDate.Week
LEVEL Month IS DDate.Month
LEVEL Quarter IS DDate.Quarter
LEVEL Year IS DDate.Year
HIERARCHY Week_rollup
( Day CHILD OF Week CHILD OF Year )
HIERARCHY Calendar_rollup
( Day CHILD OF Month CHILD OF Quarter CHILD OF Year )
ATTRIBUTE Month DETERMINES ( Monthname );
```

⇒ Hierarchie wird durch Dimension wiederhergestellt
verbessert Befehls-effizienz

Exercise
Complete with

- WeekInYear
- MonthInYear
- QuarterInYear

Optimizer:

- nimmt aggregierte Tabellen mit Query Rewrite, durch Dimensions wird auch z.B. für Jahresabfrage die aggregierte Monats-View benutzt
- Average of Average ist ein statistisches Problem → für Funktion Average wird nicht die MV benutzt

Analytische Funktionen:

- ersetzen komplexe Anfragen durch einfache Funktionen, erhöhen die Performance
- Tabellen werden nur einmal gescannt und die Daten dann gebuffert
- Summieren:
 - o SUM (...) OVER (...)
 - SUM über den kompletten gewählten Bereich
- Prozent:
 - o RATIO_TO_REPORT (...) OVER (...)
 - prozentualer Anteil über alle gewählten Spalten
- Syntax: **function(argument) OVER ([partition_by_clause] [order_by_clause [window_clause]])**
 - o Partition By: z.B. PARTITION BY ArtID → immer summieren über Artikel
 - o Order By: z.B. ORDER BY Name → Sortierung nach Name, Akkumulation in Partition
 - o Window Clause: z.B. RANGE BETWEEN 1 PRECEDING AND 1 FOLLOWING
 - Summieren über eins vorher und eins nachher, **nur erlaubt mit order by Klausel!**
 - z.B. mit ORDER BY Month → immer Addition mit einem Monat vorher & nachher
- Rang:
 - o RANK () OVER (ORDER BY SUM (price * quantity) DESC)
 - Rangliste mit Order und Desc → Ausdruck hinten wird von 1 - ... geranked

Statistic functions	Other functions	
Sum	Ration_To_Report	the ratio of a value
Count	Rank	rank in a row (1, 2, 3, 3, 5)
Avg	Denserank	rank in a row (1, 2, 3, 3, 4)
Max	Lag	the row before
Min	Lead	the row after

```
SELECT D.Month, Sum( FPrice*F.Quantity ) AS "Sales Volume",
      no function value SUM ( SUM( FPrice*F.Quantity ) ) OVER (ORDER BY D.Month) AS Accum,
      RANK ( ) OVER (ORDER BY SUM( FPrice*F.Quantity ) DESC ) AS Rank,
      Window clause SUM ( SUM( FPrice*F.Quantity ) ) OVER (ORDER BY D.Month
      RANGE BETWEEN 1 PRECEDING AND 1 FOLLOWING ) AS Accum3
FROM   Facttab F INNER JOIN DDate D USING (DSid)
WHERE  Year = 2019
GROUP BY D.Month;
ORDER BY D.Month ;
```

Umsatz von Vormonat, aktuellem und Folgemonat

Window Clauses für analytische Funktionen:

- Syntax:

{ RANGE | ROWS } BETWEEN Number PRECEDING AND Number FOLLOWING
 { RANGE | ROWS } BETWEEN CURRENT ROW AND Number FOLLOWING
 { RANGE | ROWS } BETWEEN Number PRECEDING AND CURRENT ROW
 { RANGE | ROWS } Number { PRECEDING | FOLLOWING }
- Range: zählt logisch in der ORDER BY Spalte
 - o z.B. bei Monat: wenn April nicht inbegriffen, werden nur März und Februar zusammengerechnet
- Row: zählt physisch in der ORDER BY Spalte
 - o z.B. bei Monat: wenn April nicht inbegriffen, werden März, Februar und Mai zusammengerechnet

logical

**SUM (SUM(F.Price*F.Quantity)) OVER (ORDER BY D.Month
RANGE BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS OverRange,**

physical

**SUM (SUM(F.Price*F.Quantity)) OVER (ORDER BY D.Month
ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS OverRows**

Lag und Lead:

- Syntax: {LAG|LEAD}(<argument>, <count>, <alternative>) OVER (Order by ...)
- o argument: was soll mit der Zeile gemacht werden
- o count: die wievielte Zeile vorher / nachher soll genommen werden
- o alternative: Alternative für NULL, also meistens 0
- o ORDER BY: wird benötigt, ohne geht's nicht
- Lag: vorherige Zeilen werden abgefragt
- Lead: nachfolgende Zeilen werden abgefragt

MONTH	DEC	NOV	OCT
1 201912	1.241.803,40	1.153.954,90	1.191.592,20

```

SELECT * FROM (
  SELECT D.Month, Sum( F.Price*F.Quantity ) AS Dec,
    LAG ( SUM(F.Price*F.Quantity), 1) OVER (ORDER BY D.Month) AS Nov,
    LAG( Sum(F.Price*F.Quantity), 2) OVER (ORDER BY D.Month) AS Oct
  FROM Facttab F INNER JOIN DDate D USING (DSid)
  WHERE Quarter = 20194
  GROUP BY D.Month )
WHERE Month = 201912;

```

Function	Function value	Window clause	Order by clause
Avg	necessary	allowed	allowed
Count	necessary	allowed	allowed
Sum	necessary	allowed	allowed
Max	necessary	allowed	allowed
Min	necessary	allowed	allowed
Ratio_To_Report	necessary	not allowed	not allowed
Rank	not allowed	not allowed	necessary
Dense_Rank	not allowed	not allowed	necessary
Lag	necessary	not allowed	necessary
Lead	necessary	not allowed	necessary

ordering

- The **Partition By Clause** is always allowed

Beispiel für SUM () OVER ()

```

SELECT P.ArtId, P.Name, C.Name Custname, SUM(F.Price*F.Quantity) AS "Sales Volume",
  SUM(SUM(F.Price*F.Quantity)) OVER(PARTITION BY P.ArtId ORDER BY C.Name) AS Total
FROM Facttab F INNER JOIN DProduct P USING (PSid)
  INNER JOIN DCustomer C USING (CSid)
  INNER JOIN DDate D USING (DSid)
WHERE P.Prodgroup = 'Mountainbike'
AND Month = 201901
GROUP BY P.ArtId, P.Name, C.Name;

```

ARTID	NAME	CUSTNAME	Sales Volume	TOTAL
1	100021 Mountainbike	Anton Kurz KG	4900	4900
2	100021 Mountainbike	Bike Langer	3500	8400
3	100021 Mountainbike	Biker Shop	4200	12600
4	100021 Mountainbike	Bikes a. Drives Ltd	4200	16800
5	100021 Mountainbike	Bikes Anton	1400	18200
6	100021 Mountainbike	Fred Miller	2100	20300
7	100021 Mountainbike	Henry Fallier a. Co	2800	23100
8	100021 Mountainbike	Maier Ingrid	3500	26600
9	100021 Mountainbike	Michael Kurz und Co	2800	29400
10	100021 Mountainbike	Zweirad Schwamm	3500	32900
11	100022 Mountainbike	Biciclo Luici	4200	4200
12	100022 Mountainbike	Bicycle Hammer	1400	5600
13	100022 Mountainbike	Bike Miller	1400	7000
14	100022 Mountainbike	Biker Corner	4200	11200
15	100022 Mountainbike	Fred Miller	2800	14000
16	100022 Mountainbike	Gerd Kelly	6300	20300

accumulation

the next article

Partitions:

- große Tabellen sollen in mehrere Einzelteile aufgeteilt werden
- Wichtige Eigenschaften:
 - o Transparenz: unsichtbar für den User
 - o Performanz: Antwortzeit reduzieren
 - o Concurrency: gleichzeitige Zugriffe erlauben
 - o Allocation: Tabellen auf mehrere Disks verteilen
 - o Ladezeiten: Schnellerer ETL-Prozess
- Vorgehensweisen:
 - o BY LIST (VALUES ...)
 - o BY RANGE (VALUES LESS THAN ... oder MORE THAN ...)
- Vorteile:
 - o Nicht sichtbar für den User
 - o Der Optimizer kennt die Partitionen (gute Performance)
 - o Meistens nur Zugriff auf eine oder wenige Partitionen
 - o Parallele Zugriffe möglich
 - o ETL-Prozess braucht nur die aktive Partition (z.B. von diesem Monat)
- Beispiel: in SAP BW: jeder neue ETL-Prozess erzeugt eine neue Partition

```
CREATE TABLE DCustomer (
  Csid      INTEGER      PRIMARY KEY,
  Custid    INTEGER,
  Name      VARCHAR2(25),
  Place     VARCHAR2(20),
  State     VARCHAR2(20),
  Country   VARCHAR2(15)
)
PARTITION BY LIST ( Country )
( PARTITION Europa_Centre VALUES ( 'Germany', 'Austria', 'Switzerland' ),
  PARTITION Europa_South VALUES ( 'Spain', 'Italy', 'Portugal' ),
  PARTITION America_South VALUES ( 'Brasil', 'Argentina', 'Chile' ),
  PARTITION OTHER      VALUES ( DEFAULT )
);
```

Bitmap Index:

- B-Baum: optimiert für Laufwerkzugriff, average filling degree der Blöcke von 70 – 80%
 - o Nachteil: B-Baum ist nicht gut für Abfragen mit Aggregationen
 - o Erfahrung: Index sinnvoll, wenn weniger als 7% der Daten gefragt sind
- Selektivität: Anteil der selektierten Daten an allen Daten z.B. Gender Mann = 33% (bei gleicher Verteilung)
 - o Problem: hohe Selektivität verringert die Vorteile von B-Bäumen
 - o Lösung #1: B*-Baum
 - B-Baum mit zusätzlichen Merkmalen
 - In der letzten Hierarchieebene auf den Einträgen Link zu den Daten & zum nächsten Eintrag
 - die anderen Hierarchieebenen haben nur Links auf das nächste Level
 - B*-Bäume sind „state of the art“ → implementiert in allen Datenbanken
 - Hohe Selektivität führt zu Zerfall des B*-Baums
 - o Lösung #2: Bitmap-Index
- Bitmap-Index:
 - o Vorteile: Lesezugriffe sehr schnell, Gruppenzugriffe sehr effizient
 - o Nachteil: Schreibzugriff → schon eine kleine Schreibrate von 1-2% ist ein großes Problem
 - deshalb: nur Verwendung in OLAP interessant, für OLTP nicht benutzbar
 - o z.B. bei Produktgruppen: nur 1 Bit pro Produktgruppe als Index, 6 Bits dann pro Indexeintrag (weil 6 Prodgru.)
 - o Performance:
 - Bit-Operatoren (AND, OR, NOT) sind sehr schnell
 - Hohe Effizienz und niedriger Speicherverbrauch
 - o Erfahrung: Im Data Warehouse sind Bitmap Indizes eine effizientere Alternative zu B*-Bäumen
 - o in Oracle: **CREATE BITMAP INDEX Indexname ON Table (Column list)**
 - o Optimizer benutzt automatisch Bitmap Indizes zum suchen von Ergebnissen, deutlich erhöhte Performance

```

> Queries
> Range queries:      WHERE Dsid BETWEEN 1001 AND 1400
> Aggregation:      GROUP BY Custid
> Multidimensional queries: WHERE Dsid > 1400 AND Prodgroup = ' ... '
                                AND Artid IN ( ... )
> Joins              DCustomer JOIN Facttab ON ...
> Combinations of the above

> Index useful?
> Range queries:      very useful
> Aggregation:      hardly useful
>                    → we should create aggregated cubes instead
> Multidimensional queries: very useful
> Joins              often useful or very useful, but it depends

```


Operating a Data Warehouse

ODS (Operational Data Store):

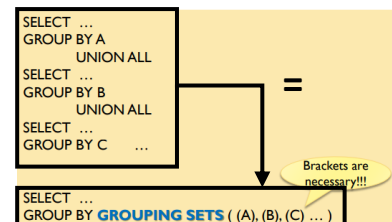
- Bereich im DW, wo zusätzliche Daten gespeichert werden (ähnliche / gleiche Struktur wie in der Quelle)
- 2 Hauptaufgaben:
 - o offene Daten (z.B. offene Bestellungen)
 - o detaillierte Daten (verschiedene Dokumente)
 - z.B. im Würfel monatliche Daten, in ODS tägliche Daten
- integriert im DW, parallel zu Würfeln
- in SAP BW: DSO parallel zu cubes, aDSO ist standard Cube in SAP BW on Hana

Name Conventions:

- für main identifier, classifying und qualifying expressions
- Abbreviation Tables für Bezeichner in verschiedenen Längen (Abkürzungen)

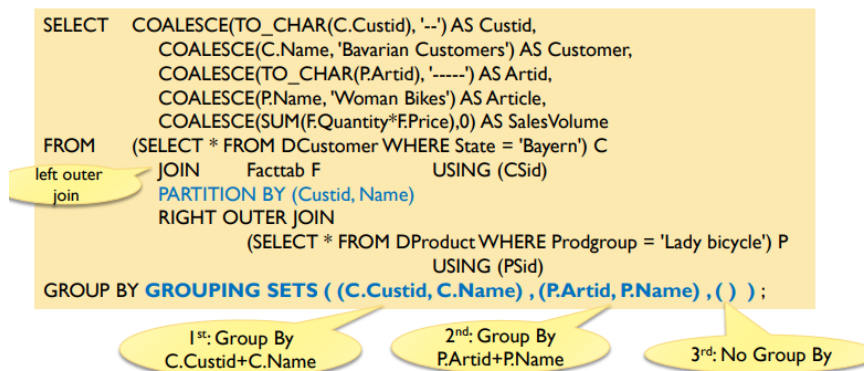
Vorbereitung von Daten:

- Aggregation entweder über Export und Vorbereitung in Excel (Pivot Tabellen), oder mit einer Datenbank (SQL)
 - o in Oracle:
 - Holzhammermethode: mit mehreren SQL-Statements und UNION ALL
 - schlechte Performance, riesige SQL-Statements, fast nicht lösbar
 - Besser: Grouping Sets, Rollup, Cube
- in SAP BW sind beide Möglichkeiten gegeben



Grouping Sets:

- Gruppierung der Daten nach bestimmten „Sets“: Code rechts macht in beiden Fällen das gleiche
- Vorteile:
 - o Gute Performanz (nur ein Scan von Fact-Table)
 - o Oracle speichert die aggregierten Daten für zukünftige Aggregationen
- Da in letzter Zeile (null)-Values ausgegeben werden → Nutzung von COALESCE
- Bsp.: Ausgabe von einzelnen Artikeln, dann von einzelnen Kunden und dann vom Umsatz zusammengerechnet



Rollup:

- Partition wie bei Grouping Sets, aber in hierarchischer Reihenfolge
- Bsp.: erst Report von Tagen, dann vom Monat, dann vom Jahr:
 - Solution


```
GROUP BY GROUPING SETS ( ( day, month, year), ( month, year), ( year), ( ) )
```
 - Equivalent


```
GROUP BY ROLLUP ( year, month, day )
```
- zuerst Gruppierung nach Jahr, dann Gruppierung nach Monat, dann Gruppierung nach Tag (von links nach rechts) → alle einzeln aufgeführten Elemente werden wieder einzeln gruppiert, zusammenfassen mit Klammern

Nach jedem Monat wird die Monatszusammenfassung ausgegeben → ORDER BY wichtig!

Cube:

- Cube-Operator zeigt alle verschiedenen Möglichkeiten von Grouping Sets:

GROUP BY GROUPING SETS ((day, month, year), (day, month), (day, year), (month, year), (day), (month), (year), ()) = **GROUP BY CUBE** (day, month, year)

Grouping:

- Abfrage: Wenn die Spalte gruppiert wurde, dann schreib Ausdruck rein
 - o CASE WHEN GROUPING (column) = 1 THEN ,...'
 - o Benennung der aktuellen Aggregation so besser möglich, z.B. Monthly Sum, Yearly Sum, ...

```
SELECT CASE WHEN GROUPING(MonthInYear) = 1 THEN 'YEARLY SUM:'
          WHEN GROUPING(C.Custid) = 1 THEN 'MONTHLY SUM:'
          ELSE '-----' END AS "Grouping",
        COALESCE (TO_CHAR(C.CustID), '---') AS Custid,
        COALESCE(C.Name, 'Bavarian Customers') AS Customer,
        COALESCE (TO_CHAR (MonthInYear), '-') AS Month,
        TO_CHAR(COALESCE(SUM(F.Quantity*F.Price),0), '999G999G990D00')
                                                    AS SalesVolume
FROM   Facttab F INNER JOIN (SELECT * FROM DDate WHERE Year = 2019) USING (DSid)
        PARTITION BY (MonthInYear)
        RIGHT OUTER JOIN
        (SELECT * FROM DCustomer WHERE State = 'Bayern') C Using (CSid)
GROUP BY ROLLUP (MonthInYear, (C.Custid, C.Name))
```

Grouping?	CUSTID	CUSTOMER	MONTH	SALESVOLUME
25	1	Biker Shop	7	44.035,50
26	3	Maier Ingrid	7	37.499,50
27	6	Bicycle Hammer	7	40.146,90
28	---	Bavarian Customers	7	122.481,90
29	1	Biker Shop	8	20.142,00
30	3	Maier Ingrid	8	57.797,00
31	6	Bicycle Hammer	8	56.701,00
32	---	Bavarian Customers	8	134.640,00
33	1	Biker Shop	9	15.447,50
34	3	Maier Ingrid	9	18.722,50
35	6	Bicycle Hammer	9	25.737,20
36	---	Bavarian Customers	9	59.907,20
37	1	Biker Shop	10	37.943,00
38	3	Maier Ingrid	10	20.210,20
39	6	Bicycle Hammer	10	46.494,60
40	---	Bavarian Customers	10	104.647,80
41	1	Biker Shop	11	17.050,00
42	3	Maier Ingrid	11	34.365,50
43	6	Bicycle Hammer	11	27.905,50
44	---	Bavarian Customers	11	79.321,00
45	1	Biker Shop	12	66.803,50
46	3	Maier Ingrid	12	54.605,90
47	6	Bicycle Hammer	12	49.720,00
48	---	Bavarian Customers	12	171.129,40
49	---	Bavarian Customers	---	1.340.559,00

- Allgemein:
 - o 2 verschiedene Möglichkeiten:

```
CASE Expression WHEN Value THEN Return_Expression [WHEN ... ]
      [ELSE Return_Expression]
END

CASE WHEN Condition THEN Return_Expression [WHEN ... ]
      [ELSE Return_Expression]
END
```

```
CASE Year WHEN 2018 THEN '2018'
          WHEN 2019 THEN '2019'
          WHEN 2020 THEN '2020'
          ELSE '2021'
END
```

```
CASE WHEN YEAR < 2020 THEN 'old'
      WHEN YEAR = 2020 THEN 'last year'
      ELSE 'now'
END
```

Beispiel für Cross Join:

```
SELECT COALESCE (TO_CHAR (salesdate), 'SUM') as Day,
        COALESCE (dproduct.name, 'SUM') as Name,
        COALESCE (color, 'SUM') as Color,
        COALESCE (SUM (f.price*quantity), 0) as Revenue
FROM facttab f
        INNER JOIN (SELECT * FROM dcustomer WHERE state = 'Hessen') USING (Csid)
        RIGHT OUTER JOIN (SELECT * FROM ddate WHERE monat = 202011) USING (Dsid)
        PARTITION BY (salesdate)
        RIGHT OUTER JOIN (SELECT * FROM dproduct WHERE prodgroup = 'mtbnbike') USING (Psid)
GROUP BY Grouping Sets ((salesdate, name, color), ( ))
```

Beispiel für LAG / LEAD Abfrage:

```
SELECT month, rev20, rev19, rev18
FROM (
    SELECT year, COALESCE(monthinyear, 0) month,
           COALESCE(sum(f.price*quantity), 0) rev20,
           LAG(COALESCE(SUM(f.price*quantity),0), 12, 0) OVER (ORDER BY year, monthinyear) rev19,
           LAG(COALESCE(SUM(f.price*quantity),0), 24, 0) OVER (ORDER BY year, monthinyear) rev18
    FROM facttab f
        INNER JOIN (SELECT * FROM dproduct WHERE name = 'mtbnbike') USING (Psid)
        RIGHT OUTER JOIN (SELECT * FROM ddate WHERE year IN (2018, 2019, 2020)) USING (Dsid)
    GROUP BY (year, monthinyear)
)
WHERE year = 2020;
```

Materialized View:

- REFRESH FAST ON COMMIT → nach jedem Commit wird ein schneller Refresh durchgeführt

Metadata:

- Informationen um Konstruktion, Wartung und Administration zu vereinfachen
 - o descriptive Information über Inhalt, Struktur, Kontext und Sinn der Daten
 - o process relational Information über das Data Processing
- gibt Information darüber, wo die Daten im DW herkommen, wie die Daten transformiert wurden, wann und wo die Daten in die Würfel geladen wurden, über die Struktur und den Inhalt der Würfel

History:

- DW wird ständig geändert, aber es gibt auch seltene Änderungen (Umzug von Kunden, neue Produktgruppe,...)
 - o Facttable (transaktionelle Daten): die betroffenen Daten werden gelöscht und neu angelegt
 - o Dimension table (master Daten): Duplikat mit Änderung wird angelegt, neue ID, neue darauf referenzierte Daten bekommen neue ID mit
- SCD: slowly changing dimensions

Date and time:

- Datum und Zeit sind separiert, Datum hat eigene Dimension Table
- Zeit wird extra gespeichert wenn sie benötigt wird (neue Dimension)
- Probleme:
 - o Wochen 1 und 53 sind nicht vollständig
 - o Zeitzonen in verschiedenen Ländern spielen eine Rolle
 - o Sommer- und Winterzeit

▶ **In practice**

- ▶ Base: Local time of the head office
- ▶ If required: Not only save date but also time of day
- ▶ Shipping: Departure and arrival time in local time

Währung:

- Geldsumme wird in lokale Währung umberechnet und dann im Data Warehouse gespeichert

Hierarchien:

- in manchen Tabellen hat die Hierarchie ein festes Limit (vgl. DDate)
 - o in manchen anderen nicht (z.B. Articlestructure)
- Bridge Tables:
 - o gibt Hierarchie (Father – Son) an und Hierarchieebenen (Levels between & lowest Level)

Father	Son	Levels between	lowest level
Germany	Germany	0	N
Germany	Bavaria	1	N
Germany	Oberpfalz	2	N
Germany	Kreis Regensburg	3	N
Germany	Regensburg	3	J
Germany	Neutraubling	4	J

Junk Dimensions:

- alle Attribute, die nicht in andere, existierende Dimensionstabellen passen, werden in einer Junk Dimension zusammengefasst
- z.B. Indikator zum analysieren und verifizieren von verkauften Produkten

Fact tables without facts:

- in manchen Würfeln existieren keine Fakttabellen
- z.B. bei einer Auktion wird nichts verkauft, trotzdem sind Angebote interessant

Protocoll Dimension:

- enthält alle Ladeprozesse für die Würfel
- enthält alle durchgeführten Transaktionen

in SAP BW immer 3 gegebene Dimension Tables:

- o Time: für Zeit- und Datumswerte
- o Unit: für Währungs- und Messeinheiten
- o Data Package: als Protokolldimension für das Laden von packages

SAP BW on Hana

SAP HANA:

- ERP = system supporting enterprise resource planning
- Kompatibel zu fast jedem enterprise system durch die Nutzung von über 1000 Regulierungsschrauben

SAP BW:

- Data Warehouse von SAP mit SAP GUI für den Client
- Applikationsserver und relationale Datenbank in Speicherdatenbank (SAP BW on HANA)













Areas in SAP:








- ▶ **Modeling:** Create master data, cubes, ETL processes
- ▶ Administration: Monitors for monitoring loading processes
State overview of Data Store Objects
- ▶ Transportconnection: Tracing transport requests
- ▶ Documents: Internal documentation of objects
- ▶ BI Content:
- ▶ Translation:
- ▶ Metadata Repository: Overview of active objects
Overview of BW

Modeling:

- ▶ **Data-Flows:** Administration of ETL data flow
- ▶ **InfoProvider:** Administration of cubes and DSO
- ▶ InfoObjects: Administration of Master Data
Defining of characteristics and key figures
- ▶ InfoSources: in BW7: not important; Staging Area
- ▶ **DataSources:** Entry for external data
Administration of staging area (PSA)
- ▶ Source System: SAP, Databases, Files

Bezeichnungen:

	Key figure	e.g. sales volume, gain
	Characteristic	e.g. customer, product
	Info Cube	Cube, prepared for OLAP queries
	Dimension	Dimension of a cube
	Hierarchies	Hierarchies within master data
	Maintain Master Data	-
	Text	Text within master data
	Info Object	Container of key figures and characteristics
	Data Source	Entry area for source data
	Info Source	Definition area for loading processes
	Source System	e.g. SAP ERP, databases, files
	Info Area	Directory, to define cubes and master data

	Data Store Object	DSO = ODS, not prepared for queries
	Info Provider	Container for InfoCubes and DSO
	SID	Surrogate ID
	Info Object Catalog	Catalog for key figures and characteristics
	Persistent Staging Area	PSA, Staging area of SAP BW
	Transformation	Definition of a transformation
	Load process	Definition of the load process

Vorgehensweise:

- Anfang:
 - o Info Object wird erstellt, definieren von Info Object Catalog, Key figures und Characteristics
 - o danach Info Provider, definieren von Master Data, Info Cube und DSO (Data Store Object)
 - o als letztes zur Data Source, wo input prozesse, data transfer prozesse und transformationsprozesse definiert werden

SAP HANA Studio:

- definieren von Strukturen im DW, vor allem die benötigten Objekte und aDSO (NUR in Hana Studio)
- Queries für die Daten

SAP Workbench:

- Objekte von HANA sind sichtbar
- externen Inhalt in die Objekte laden (externer Inhalt ist keine Datenbank, sondern CSV-Datei)
- für den Ladeprozess kein Unterschied ob Datenbank oder Dateien
- kompletter ETL-Prozess in der Workbench

Transaction RSRT:

- RSRT zum analysieren und debuggen der Abfragen in SAP

InfoObject: Fokus auf business requirements

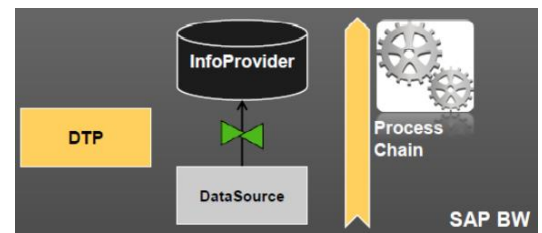
- Datenstruktur wird auf Infoobjects gebaut, technisch wie eine Datenbank mit Eigenschaften
- am wichtigsten: Characteristics und Key Figures
 - o Characteristics haben zusätzlich zu Key und Beschreibung weitere Details
 - z.B. „Customer“ hat noch „Location“, „Sales Organisation“, „Country“ ...
 - oft Attribute die eine Hierarchie formen (Hierarchie wird nicht von selbst angezeigt, kann einzeln angezeigt werden)
- InfoObjects sind unique in SAP BW, bauen Datenstruktur für Analyseanforderungen
 - o conceptionally unique: Semantik ist einzigartig systemweit definiert
 - o logically unique: Datentyp, Value range und field length sind einzigartig systemweit definiert
- Business Transaction: Datensatz, der aus einer Kombination von Characteristics und key figures besteht

InfoArea:

- Verzeichnis zu Definition von Cubes und Master Data

InfoProviders:

- Datencontainer für DSOs und Cubes, speichern Daten physisch
- physische Daten: aDSO (advanced data store objects) → Daten werden optimiert
- virtuelle Daten: composite providers
- hierarchische Darstellung des gesamten Ladeprozesses von der Quelle bis zum Cube



InfoObjects:

- zu Administration der Master Data, Container für Characteristics und Key Figures
- InfoObjectCatalog: Tabellenspalten für Characteristics und Key Figures

Advanced Data Store Objects:

- können aus InfoObjects oder Datenbankfeldern bestehen
- besteht maximal aus 3 Tabellen:
 - o Inbound Table, Change Log und Active Data
 - o Daten werden immer auf inbound table geschrieben, im Modus „Activate Data“ werden sie in die Active Data Tabelle geschrieben (während Kompressionsprozess)
 - o Direct update: Daten werden direkt auf Active Data Tabelle geschrieben
 - o Plannin Mode: Objekt kann zum planen genutzt werden

Composite Provider:

- persistiert keine Daten, kann Daten von verschiedenen InfoProviders zusammenführen
 - o UNION und verschiedenen JOIN Typen werden benutzt
- wird in Queries verwendet, eine Art Sicht auf das aDSO Objekt
- Zwischenschicht zwischen aDSO und Queries

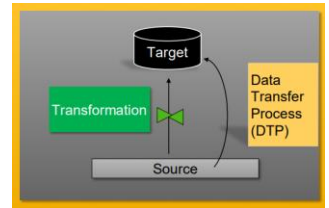
Conversion Routine:

- transformiert externe Daten in internes Format, z.B. Datum YYYYMMDD zu DD.MM.YYYY
- ALPHA Conversion Routine: nimmt Leerzeichen & 0er am Anfang raus („_1000“ wird zu „1000“)

ETL-Prozess in SAP:

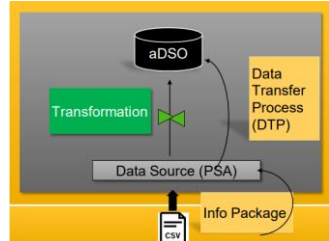
Datenfluss:

- Data Source: interface object zum source system, repräsentiert die source data structure, besteht aus Feldern und adapter settings
- Transformation: transformiert die Daten nach den definierten Regeln
- Data Transfer Process (DTP): triggered Datentransfer von Quelle zu Ziel
- Process Chain: Sequenz von Prozessen, die in einer bestimmten Reihenfolge getriggert werden



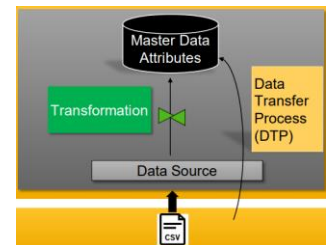
Transaktionsdaten:

- Die Daten werden von der Quelldatei zur PSA durch ein **InfoPackage** transferiert
- Data Source hat Persistenz (PSA / Persistent Staging Area)
 - o temporäre Tabelle, die Daten im data acquisition layer speichert, bis sie in die oberen Layer des DW transferiert werden
 - o für Transformation, Umwandlung und Transfer der Daten in Cube zuständig



Master Data:

- 2-Schritt-Prozess: Attribute und Beschreibungen (Text) müssen geladen werden
- Data Transfer Process (DTP) kann Datei übertragen, ohne PSA zu betreten (bypassing PSA)
- alle Kennzahlen (Key Figures) und alle Merkmale (Characteristics)
- Master Data Tables gehören zu InfoObjects, wird separat zu transaktionellen Daten geladen
- können Hierarchien und komplexe Strukturen enthalten



Full vs. Delta Data Upload:

- Full: Alle Daten aus der Data Source extrahieren
- Delta: Nur Daten aus Data Source extrahieren, die sich seit dem letzten Update geändert haben

Data Transfer Process:

- Monitor: in InfoProvider Management & DataTransferProcess Management integriert
 - o Error Stack wird im Monitor gezeigt
- Error Handling:
 - o Valid Records Update, No Reporting at all (FAIL)
 - o No Update, No Reporting (FAIL)
 - o Valid Records Update, No Reporting (FAIL)
 - o Valid Records Update, Reporting Possible (OK)

DSOs → siehe S. 9 oben

Cube:

- aufbereitete Daten für OLAP-Queries
- 3 MUSS-Dimensionen: Data Package, Time, Unit + Modelling Dimension
- extended Star Model: keine Master Data in Dimensionstabellen, sondern nur SID auf Master Data

Transformation von Währungen & Messeinheiten:

- automatisches Hinzufügen von InfoObjects (0Currency, 0Baseum) → alte Daten aus Data Source müssen gelöscht werden, bevor DataStore Objekt aktiviert wird (sonst inkonsistent)
- alle Währungskonvertierungen werden in einem „conversion type“ gespeichert, kann zur query time konvertiert werden

Selections & Filter:

- mit Selections können Filter direkt an eine Key Figure gebunden werden
 - o Einschränkung der Kennzahl
- Filter schränken eine komplette Query ein

Navigationsattribut:

- ein Attribut eines Merkmals, dass in Abfragen aktiv verwendet werden kann
- wird definiert in einem InfoObject (bzw. in einem Merkmal) und entsprechend gekennzeichnet
- wird aktiviert im Composite Provider

Query Designer:

- vorgegebene und frei auswählbare Characteristics:
 - o vorgegeben: in Rows und Columns ablegen, werden in Query beim Start sofort verwendet
 - o frei wählbar: in „free Characteristics“ ablegen, **können** in Query verwendet werden

Query:

- liest Daten von einem InfoProvider (CompositeProvider), der auf einem aDSO basiert
- Definiert ein subset von den Characteristics und KeyFigures des Infor Providers und filtert diese
- ist ein multidimensionales Objekt
- wir definieren eine Start View: welche Elemente sind auf Spalten & Zeilen
- kann Formeln, Konditionen, Exceptions, Hierarchien und mehr enthalten
 - o Formeln werden lokal in der Query als Teil der KeyFigure Struktur definiert → nicht wiederverwendbar
 - o calculated key figures werden in der query erstellt, aber separat gespeichert und können in anderen queries auf dem selben Inforprovider wiederverwendet werden
- die Metadaten paramterize den Analytic Manager des BW (also die OLAP Engine)
- dediziertes Tool um die Querys wiederverwendbar zu designen

